1 (1.5 ptos) Dada la gramática que permite generar cadenas de signos + y – con el mismo número de signos de cada tipo:

`S → + S - | + -`

Se pide: a) Construye la tabla de análisis sintáctico descendente. Transfórma la gramática si es necesario para que sea LI to de predicción y de sincronización. c). Indica el contenido de la p cadena ++--.

2 (1.5 ptos)

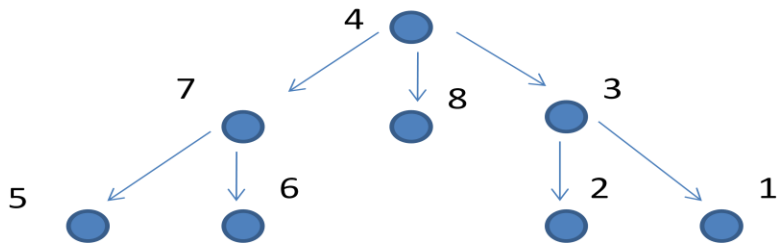
- Justifica que la sigu gua y construye una gramática no ambigua equivalente:

  `A → b | A A`

- Dada la siguiente gramática ambigua que permite generar expresiones numéricas, construye una gramática LL(1) de forma que el operador suma tenga menor prioridad que el operador potencia, que el operador + sea asociativo a izquierdas y ^ a derechas.

  `E → n | E + E | E ^ E`

- Dado el siguiente gráfico como el esquema del árbol de derivación de cierto programa según cierta gramática de atributos. Considera que los números anotados en sus nodos se corresponden con un orden compatible con la evaluación de los atributos involucrados en el análisis semántico de ese programa. ¿Cuáles de las siguientes afirmaciones son correctas? a) El orden sugerido por la evaluación de los atributos es compatible con un analizador sintáctico ascendente LR; b) El orden para la evaluación de los atributos es compatible con un analizador sintáctico descendente LL. c) Un analizador sintáctico ascendente podría incluir la siguiente secuencia en el recorrido de los nodos {5, 6, 7, 8, 2, 1, 3, 4}. Justifica la respuesta.



3 (3 ptos) Dada la siguiente gramática de operadores del lenguaje Ada:

```
S → loop S until C | S ; S | other
C → id
```

Se pide: a) Construye la tabla de precedencia de operadores. b) Implementa un sencillo mecanismo de recuperación de errores a nivel de frase. c). Haz la traza para la cadena `loop other other until id`. Indica el mensaje de error y cómo se realiza la recuperación.

4 (3 ptos) Se desea diseñar un protocolo de comunicación entre dos ordenadores. Con el fin de incrementar la seguridad, se requiere que la secuencias que se transmiten estén compuestas de sub-secuencias (bloques) que tienen entre ellas la misma longitud. Los bloques están separados unos de otros por un *. Podéis usar una variable global de tipo entero para almacenar el tamaño que los bloques deben tener y que se calcula en la derivación del no-terminal I. Los bloques se generan a partir del no-terminal B.

```
X → I | X * B
B → 0 B | 1 B | 0 | 1
I → 0 I | 1 I | 0 | 1
```

Ejemplo: `001*111*010` es correcta, `001*11*010` no es correcta.

Se pide: a) Diseña una gramática de atributos que determine si una secuencia es correcta. b) Indica de qué tipo de atributo/s se trata; c) Implementa una función recursiva para su evaluación; d). Construye el árbol y evalúa el atributo/s para la entrada `001*110`.

5 (4 ptos) En el lenguaje Ada, existe una construcción de programación que es un bucle simple combinado con una instrucción `exit`, que permite implementar bucles donde la condición de salida se evalúa al principio, en el medio o al final de cada iteración. La sentencia `exit` hace abandonar el bucle.

```
S → loop S* exit when C S* end loop
```

Se pide: a) Dibuja el diagrama de flujo. b). Indica la forma del árbol que construirías. c) Implementa la función genera código. d). Construye el árbol de análisis sintáctico e indica la lista de cuádruplos para este ejemplo.

```
loop
     a = a + 1;
     print a;
     exit when a > b;
     b = b - 1;
     print b;
end loop;
```